5
Arquitetura do AulaNet 3.0

Conforme apresentado no Capitulo 1, a arquitetura do AulaNet 3.0 pode ser vista através de duas perspectivas: a da arquitetura de aplicação e a da arquitetura técnica. A arquitetura de aplicação apresenta uma visão mais alto nível. A estrutura lógica do sistema é descrita como uma coleção de componentes relacionados e os tipos e operações obtidos na especificação são distribuídos através dos componentes. Já a arquitetura técnica apresenta uma visão mais baixo nível e inclui as partes do sistema independentes do domínio como a infra-estrutura de comunicação de componentes (ex. CORBA ou Java/RMI), a plataforma de hardware e a plataforma de software (D'Souza & Wills, 1998).

Este Capítulo retoma a especificação da arquitetura do AulaNet 3.0 descrita no Capítulo 1, para em seguida incrementá-la através do acréscimo dos frameworks de infra-estrutura analisados nos capítulos 3 e 4. É visto também como a arquitetura é adaptada para possibilitar serviços a dispositivos móveis como PDAs (*Personal Digital Assistant*) e, por fim, é visto como a arquitetura possibilita a agregação de outros frameworks, tomando como exemplo o framework de agentes de software Jade (2005) e uma prova de conceito envolvendo um dispositivo móvel.

5.1.
Elementos Principais da Arquitetura do AulaNet 3.0

A arquitetura de aplicação do AulaNet 3.0 possui 2 níveis de componentização: o de serviços e o de componentes de colaboração. Os serviços podem ser plugados e desplugados de forma a montar um ambiente customizado para cada grupo (Gerosa et al., 2005). O gerenciamento dos componentes é realizado através de frameworks de componentes, conforme a Figura 5.1 mostra.

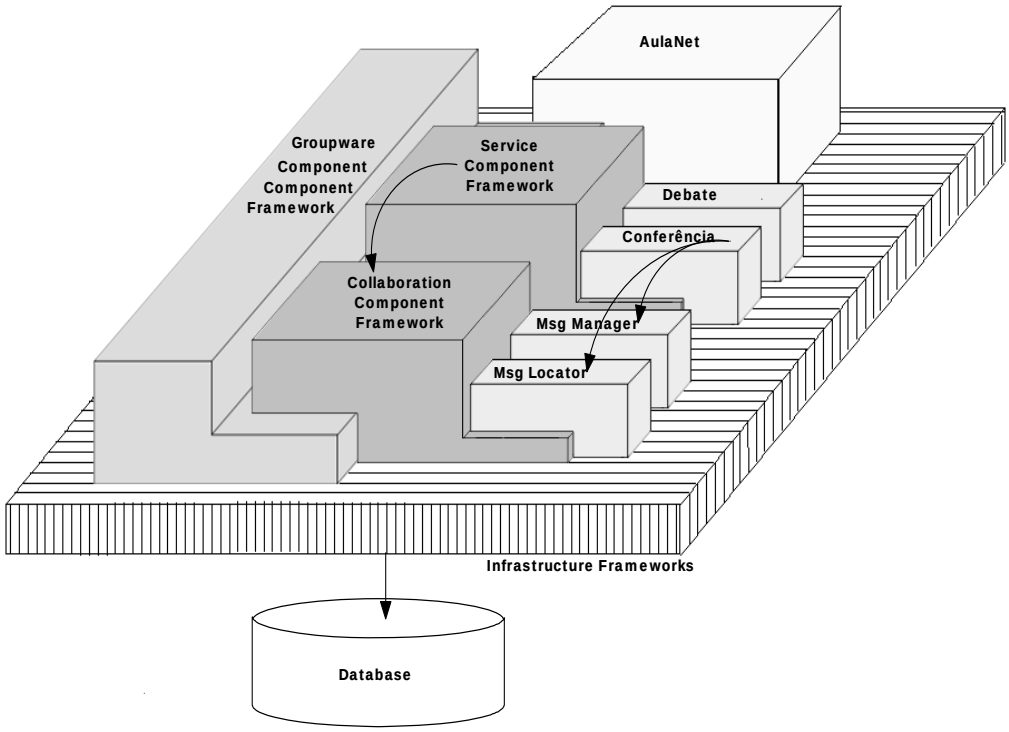

Figura 5.1 - Arquitetura de Aplicação do AulaNet 3.0 (Gerosa, 2006)
Apresentada na Seção 1.3.

O Service Component Framework provê os contratos que os serviços de groupware, por exemplo, o Debate e a Conferência, devem implementar para serem plugados na arquitetura do AulaNet. Já o Collaboration Component Framework estabelece os contratos para criar componentes 3C com os quais os serviços de groupware são criados. Por exemplo, o serviço Conferência é criado utilizando os componentes 3C gerenciador de mensagens, localizador de mensagens, etc. (Gerosa, 2006).

Os componentes são implementados segundo a arquitetura técnica que usa uma abordagem em três camadas e também o padrão MVC (Fowler, 2002). O diagrama esquematizado na Figura 5.2 mostra a arquitetura técnica do AulaNet 3.0, baseada na arquitetura de POJOs descrita em Johnson (2002, 2004). As setas indicam o fluxo de controle da aplicação, retângulos representam classes e círculos representam interfaces. Linhas pontilhadas representam a divisão entre as camadas.

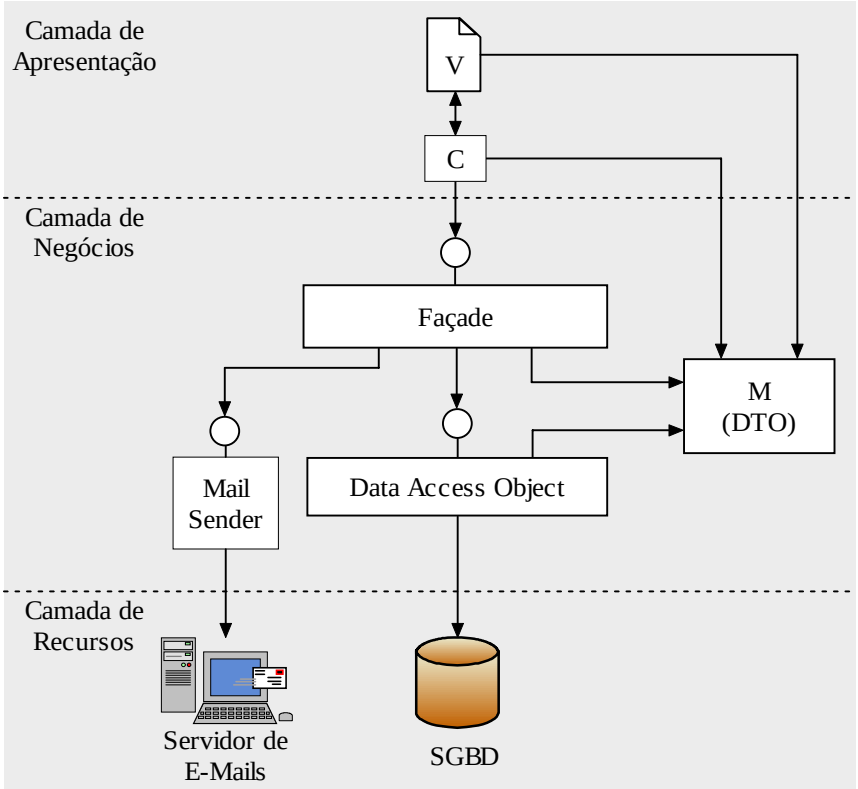

Figura 5.2 – Arquitetura Técnica do AulaNet 3.0

Apresentada na Seção 1.3.

A camada de recursos relaciona os recursos externos necessários para a execução da aplicação. Na arquitetura do AulaNet 3.0 são usados um sistema de gerenciamento de banco de dados relacional (SGBD) e um servidor de e-mails.

A lógica da aplicação é implementada na camada de negócios. As classes do modelo (M do MVC) realizam o padrão de projetos *Data Transfer Object* (Fowler, 2002) e são usadas para transportar os dados das entidades de negócio entre camadas. O acesso à base de dados é encapsulado através de classes que realizam o padrão de projetos *Data Access Objects* (DAO) (Alur et al., 2001), desta forma a maneira de persistir o modelo pode variar trocando componentes, sem que seja preciso reescrever o código cliente. O componente Mail Sender, de forma similar ao DAO, encapsula o acesso ao servidor de e-mails. A lógica de negócios é exposta para a camada de apresentação através de um *Façade*, que provê uma interface única para acesso às funcionalidades de um serviço (Gamma et al., 1995).

Os serviços plugados no Service Component Framework são criados com um único *Façade*, que expõe as operações deste serviço para a camada de apresentação. Os componentes de colaboração plugados no Collaboration Component Framework por sua vez, podem utilizar vários DTOs e DAOs, dependendo da complexidade do componente. Estes componentes podem ainda usar "código cola" (Szyperski et al., 1997) e adaptadores (D'Souza & Wills, 1998) para possibilitar a integração com componentes e outros sistemas que não são compatíveis por construção.

A camada de apresentação, que no caso de aplicações voltadas para a internet também costuma ser chamada de camada web, expõe a lógica de negócios ao usuário e possibilita a interação do usuário com a aplicação. Na arquitetura do AulaNet 3.0, a camada web é composta pelo controlador, o C do MVC além de páginas JSP, que correspondem ao V do MVC. O controlador chama os métodos do *Façade*, acessando a camada de negócios. Os DTOs resultantes de operações são passados a visão que exibe suas informações ao usuário.

Esta seção reviu a arquitetura definida para o AulaNet 3.0, do ponto de vista da arquitetura técnica e de aplicação. Ao longo desta dissertação frameworks de infra-estrutura foram acrescentados a esta arquitetura, provendo uma base de serviços que tratam aspectos como persistência de dados, gerenciamento de transações entre outros. Na próxima seção a arquitetura do AulaNet 3.0 é reformulada através do acréscimo destes frameworks.

5.2. A Arquitetura do AulaNet 3.0 e os Frameworks de Infra-Estrutura

Spring e Hibernate formam a base com os frameworks de infra-estrutura que oferecem suporte ao desenvolvimento de componentes de negócios. Eles provêm serviços de infra-estrutura, como por exemplo, persistência de dados e gerenciamento de transações para os componentes desenvolvidos com o Service Component Framework e o Collaboration Component Framework. A Figura 5.3 mostra o novo diagrama da arquitetura de aplicação do AulaNet 3.0 após a incorporação dos frameworks de infra-estrutura.

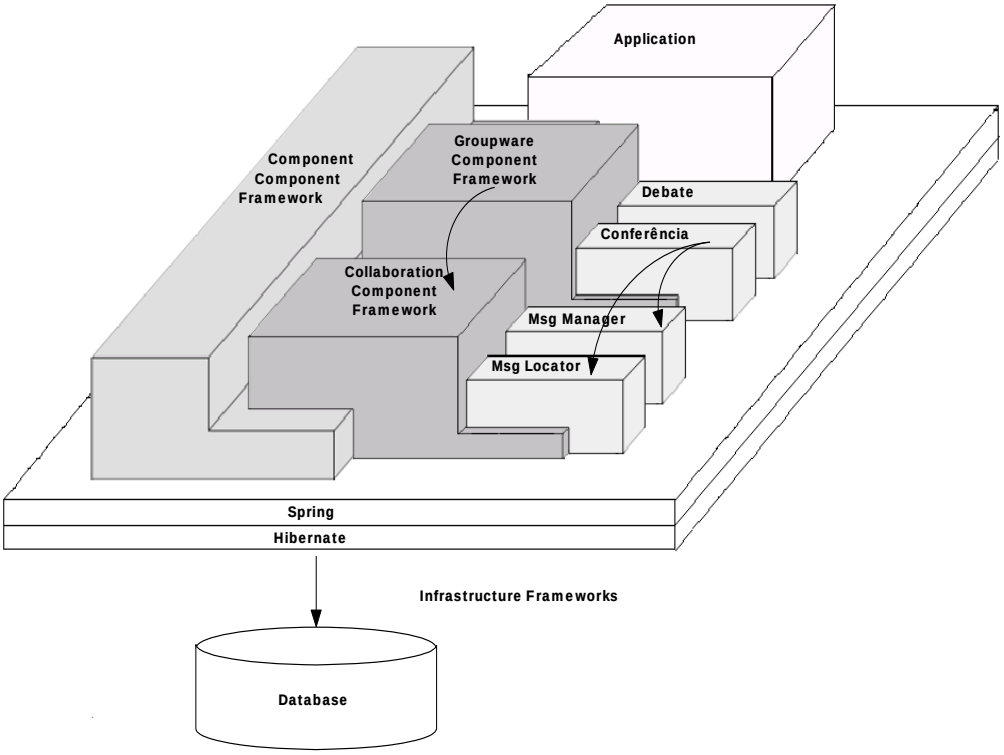

Figura 5.3 - Arquitetura de Aplicação do AulaNet 3.0 (Gerosa, 2006)

O JavaServer Faces não é mostrado no diagrama da arquitetura de aplicação pois este não contempla a camada de apresentação. A Figura 5.4 exibe o diagrama da arquitetura técnica, com os frameworks de infra-estrutura.

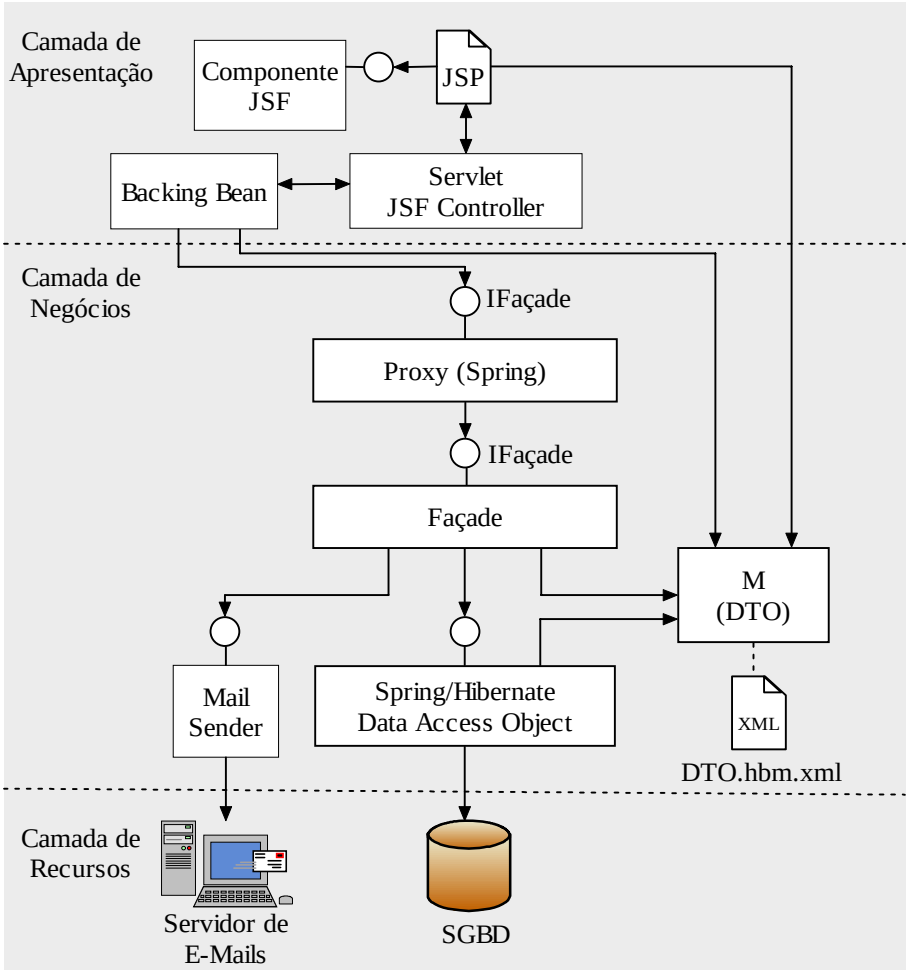

Figura 5.4 - Arquitetura Técnica do AulaNet 3.0 com Frameworks.

Na camada de apresentação, a visão é construída com páginas JSP compostas por componentes JSF. Estes podem ser componentes de validação, de conversão de dados e de interface com o usuário. O Servlet JSF Controller é usado como controlador do MVC. Ao receber uma requisição, o controlador chama o *backing bean*, que acessa a camada de negócios.

A entrada para a camada de apresentação é feita através do *Proxy* (Gamma et al., 1995) do Spring. Como o *Proxy* implementa a mesma interface do *Façade* (inteface IFaçade), o código cliente não precisa ser modificado para usar o *Proxy* em vez da implementação direta do *Façade*. O *Proxy* do Spring acrescenta serviços de controle de transações e gerenciamento de segurança. Para cada DTO representando uma entidade de negócios, um arquivo de mapeamento do Hibernate é gerado. Este arquivo possui os metadados de mapeamento objeto/relacional que possibilitam que o Hibernate realize a ponte entre os

paradigmas objeto e relacional. Por fim, os DAOs são construídos usando tanto o Hibernate, que possibilita o mapeamento de objetos em base de dados relacionais, quanto o Spring, que gerencia a abertura e o fechamento de sessões do Hibernate e oferece métodos template, simplificando o desenvolvimento e reduzindo a ocorrência de erros. O Spring também atua na configuração das dependências dos componentes da camada de negócio.

Como é mostrado nesta seção, a arquitetura de POJOs do AulaNet 3.0 possibilita o acréscimo de frameworks de infra-estrutura. Estes frameworks por sua vez, possibilitam que o desenvolvedor de groupware concentre-se em seu domínio, ou seja, groupware, deixando aspectos de infra-estrutura para outros frameworks especialistas. A arquitetura do AulaNet 3.0 também provê suporte a outros tipos de aplicações clientes, por exemplo, clientes móveis, como é visto a seguir.

5.3. Arquitetura Técnica e Mobilidade

Com o crescimento da utilização de equipamentos móveis e redes sem fio, o potencial de uso de serviços tradicionais, como a navegação na web e e-mail aumentou bem como serviços não tradicionais e específicos de aplicações móveis, como aqueles que fazem uso da informação física do usuário. Através da adoção destas tecnologias espera-se que seja possível comunicar-se e ter acesso a informações e serviços em qualquer lugar e a qualquer instante. Neste contexto, a educação deverá incluir soluções que façam uso destes recursos (Filippo et al., 2005a).

Com o objetivo de investigar mecanismos para aumentar a colaboração na aprendizagem através do uso de equipamentos móveis, iniciou-se o desenvolvimento de uma extensão do ambiente AulaNet específica para este fim, denominada AulaNetM (Filippo et al., 2005a). A versão atual do AulaNetM integra-se ao AulaNet 2.1 no nível de dados. A arquitetura do AulaNet 3.0 foi projetada para possibilitar integração no nível de serviços, aumentando assim a reutilização entre os dois projetos. Há duas formas de possibilitar a integração da arquitetura do AulaNet 3.0 no nível de serviços com o AulaNetM: através da reposição da camada de apresentação e da exposição de serviços remotamente.

A técnica da reposição da camada de apresentação consiste em reaproveitar todo o código da camada de negócios e combiná-lo com uma nova camada de apresentação, desenvolvida especificamente para o dispositivo móvel. A aplicação resultante é executada independentemente do AulaNet 3.0 e serve aplicações clientes magras, ou seja, clientes que rodam em navegadores HTML no caso de PDAs ou WML no caso de celulares. Esta aplicação pode ser instalada no mesmo servidor do AulaNet (em um contexto diferente) ou em outro servidor.

Opcionalmente, são acrescentados serviços específicos para clientes móveis, como serviços sensíveis a contexto ou a localização. Os projetistas da aplicação móvel têm controle limitado sobre o navegador utilizado no dispositivo e quase sempre não há suporte computacional para recuperar dados como a localização do PDA e o nível de carga da bateria. Torna-se necessário então que a aplicação questione o usuário sobre estas informações para poder prover estes serviços. A Figura 5.5 exibe a arquitetura do AulaNetM integrada ao AulaNet 3.0, usando esta técnica.

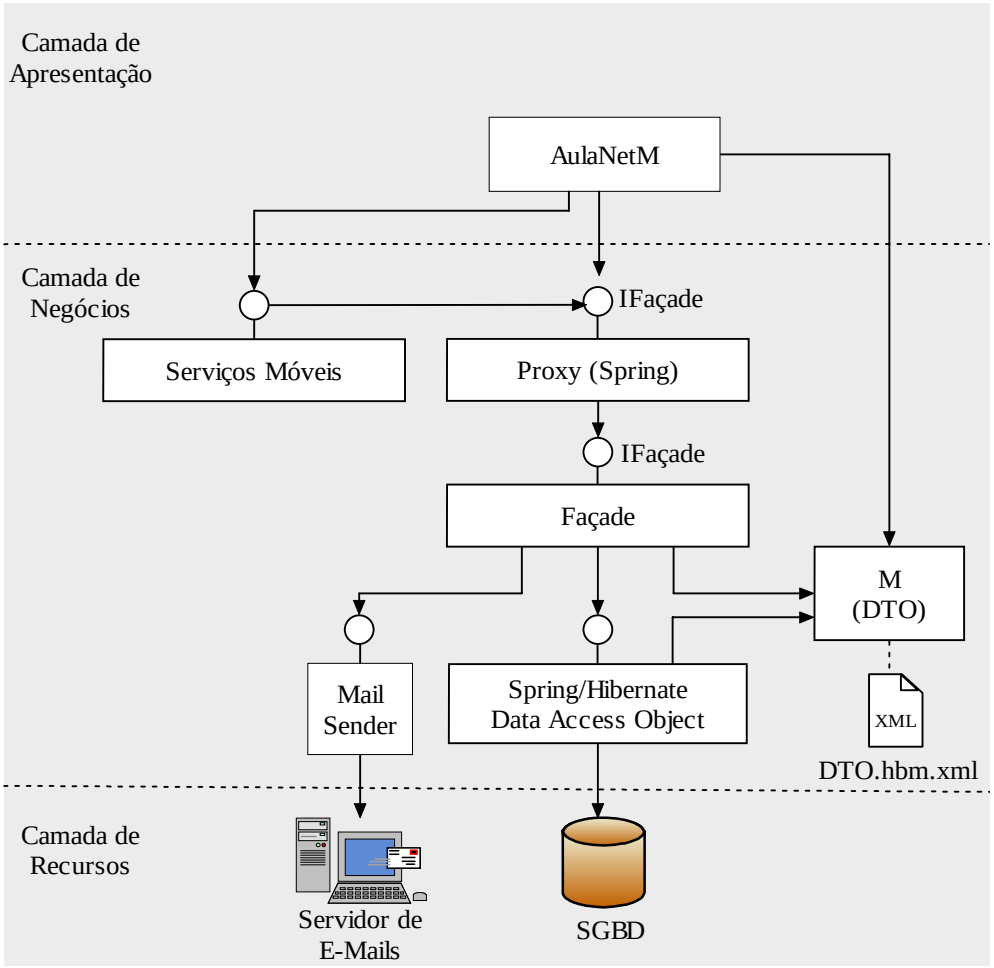

Figura 5.5 – Arquitetura Técnica da Integração AulaNet 3.0 com AulaNetM por Reposição da Camada de Apresentação

A não ser pela adição dos serviços móveis, que é opcional, a camada de negócios é a mesma usada na arquitetura do AulaNet 3.0. A camada de apresentação, assim como os serviços específicos de mobilidade, são implementadas pelos projetistas do AulaNetM. O uso de MVC não é obrigatório, mas é recomendado.

Esta forma de integração atua em nível de serviço, mas em instâncias diferentes da camada de negócios, pois a camada de negócios precisa ser implantada no servidor do AulaNet 3.0 e do AulaNetM. A principal vantagem desta técnica é que as duas aplicações usam o mesmo código, da mesma forma com a exceção dos serviços específicos relativos à mobilidade. Equipes trabalhando em um projeto tendem a encontrar mais facilidade para mudar para outro já que estão habituadas as mesmas interfaces da camada de negócios. A

principal desvantagem é que as informações do dispositivo como estado da conexão, carga da bateria, localização, etc., não podem ser recuperadas de forma transparente para o usuário. Além disso, é despendido um esforço extra para manter os dois projetos sincronizados, já que quando ocorrerem mudanças na camada de negócio, é preciso implantar a nova versão nos servidores do AulaNetM e AulaNet 3.0.

A técnica da exposição de serviços remotos consiste em expor os serviços do AulaNet 3.0 remotamente para outras aplicações clientes. Esta técnica pode ser aplicada tanto para clientes magros quanto para clientes gordos, ou seja, clientes escritos em J2ME (2005), SuperWaba (2005) ou qualquer outra tecnologia usada para programar em dispositivos móveis.

O desenvolvedor tem total controle sobre clientes gordos, sendo assim, estes tipos de clientes têm acesso a informações de contexto como localização, nível da bateria do dispositivo e estado da conexão, desde que a tecnologia utilizada para o desenvolvimento do cliente ofereça suporte a estas informações. Contudo, eles precisam ser instalados diretamente no dispositivo. A Figura 5.6 exibe a arquitetura do AulaNetM integrada ao AulaNet 3.0, usando a técnica de exposição de serviços remotos.

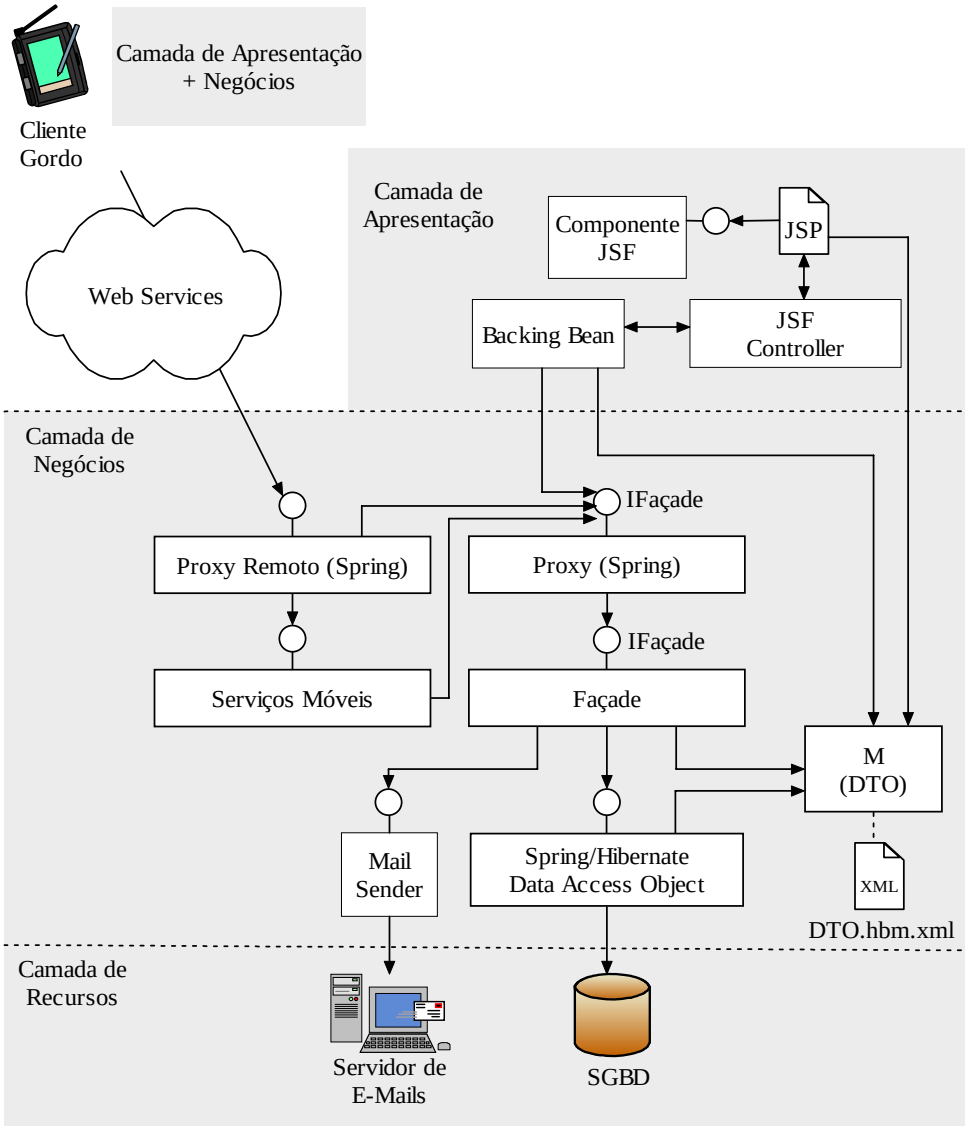

Figura 5.6 - Arquitetura Técnica da Integração AulaNet 3.0 com AulaNetM por Exposição de Serviços Remotos

Um cliente gordo tem a sua camada de apresentação implementada com tecnologias como o J2ME, mas também pode ter uma camada de negócios extra, onde há serviços independentes do acesso ao AulaNet 3.0, como um serviço despertador. Através de Serviços Web o cliente acessa a lógica de negócios do AulaNet 3.0. No caso de clientes magros, não mostrados na figura acima, a camada de apresentação é implementada utilizando HTML ou WML através das tecnologias JSP e Servlets, que acessam a camada de negócios do AulaNet 3.0 através de Serviços Web.

Como é visto no capítulo 3, o Spring possibilita a exposição de serviços de diversas formas. A forma escolhida para integração com AulaNetM a princípio é por Serviços Web (Web Services), pois possibilitam a integração com aplicações clientes escritos em várias linguagens de programação além do Java. O Spring também possibilita que outros *Proxys* remotos sejam acrescentados, dando suporte ao uso de outros protocolos, como o RMI (RMI-IIOP, 2005), o Hessian/Burlap (Caucho, 2005) e o HttpInvoker (Spring, 2005). Vale lembrar que o uso de *Proxys* no Spring é feito através de configurações no descritor da aplicação, sem que seja necessário alterar o código dos componentes.

Os *Proxys* remotos recebem as chamadas remotas e as encaminham para os métodos da camada de negócios ou para os componentes que provêm serviços específicos de mobilidade. A arquitetura do AulaNet 3.0 não sofre alteração substancial com o acréscimo dos serviços remotos.

A principal vantagem do uso da técnica de exposição de serviços remotos é que ela dá suporte à construção de clientes gordos, que têm mais controle sobre o dispositivo móvel. A principal desvantagem é que chamadas remotas de métodos são mais caras que chamadas locais, tanto no que diz respeito ao desempenho quanto à complexidade adicionada a programação. Esta técnica pode ser usada tanto com clientes gordos quanto com clientes magros, mas em se tratando de clientes magros, é mais vantajoso usar a técnica da reposição da camada de apresentação que utilizam chamadas locais.

Apesar das promessas de ensino e aprendizagem em qualquer lugar e a qualquer instante, a educação através de dispositivos móveis (*m-learning*) está sujeita a alguns desafios: a conexão é intermitente e lenta; pouco poder de processamento de dispositivos móveis; fontes de energias limitadas que de tempos em tempos necessitam ser recarregadas; adaptação das aplicações a um determinado aprendiz, ao contexto em que ele esta situado, à sua localização, etc. Estes desafios são inerentes às características dos dispositivos móveis e podem ser considerados também como desafios para outros tipos de groupware executados em dispositivos móveis. Sistemas de agentes inteligentes têm um grande potencial para solucionar estes desafios (Kinshuk & Lin, 2004).

5.4. Arquitetura Técnica e Outros Frameworks: Jade

Apenas os frameworks Spring, Hibernate e JSF foram selecionados para fazer parte da base de serviços de infra-estrutura da arquitetura do AulaNet 3.0, contudo esta pode ser estendida para incorporar outros frameworks. Como estudo de caso esta seção mostra como a arquitetura do AulaNet 3.0 foi estendida para incorporar o framework de desenvolvimento de agentes Jade (Bellifemine et al., 2003).

5.4.1. Agentes de Software

O termo agente tem sido amplamente usado por pesquisadores em áreas similares, o que torna difícil obter uma definição precisa e amplamente aceita. Segundo Wooldridge & Jennings (1995) agentes pode ser definido seguindo uma noção fraca ou forte.

Segundo a definição fraca, agente é definido como um sistema computacional com as seguintes propriedades: autonomia, agentes operam sem a intervenção de seres humanos e possuem controle sobre suas ações e estado interno; habilidade social, agentes interagem com outros agentes e possivelmente com seres humanos através de uma linguagem de comunicação de agentes (*agent-communication language* - ACL); reatividade, agentes percebem seu ambiente (que pode ser o ambiente físico, a internet, a interface gráfica com o usuário) e são capazes de reagirem a mudanças neste ambiente; e pró-atividade, agentes tomam a iniciativa de agem, sem receber ordem externa.

Já a noção forte, usada principalmente por pesquisadores na área de inteligência artificial, acrescenta às propriedades descritas na noção fraca características que geralmente são associadas a seres humanos. Por exemplo, Shoham (1993) adiciona as noções de conhecimento, crenças, intenções e obrigações aos agentes.

Para esta dissertação, a definição fraca de Wooldridge & Jennings (1995) é adotada. É considerado também que agentes possuem a característica da mobilidade, que possibilita que agentes se movam pela rede (White, 1994).

Pesquisas de sistemas multi-agentes (*Multi Agent Systems* - MAS) seguem duas linhas (Torres & Lucena, 2001). A primeira considera agentes como elementos de primeira ordem. Pesquisadores desta linha vêem MAS como o elemento fundamental de uma nova engenharia de software para a qual devem ser desenvolvidas novas linguagens, metodologias e técnicas de modelagem. Já pesquisadores da segunda consideram agentes uma abstração que deve ser usada para complementar o paradigma da orientação a objetos. Pretende-se seguir a segunda linha de pesquisas ao acrescentar um framework de agentes ao AulaNet.

5.4.2. Aplicações de Sistemas Multi-Agentes

Como é visto na seção 5.3, o *m-learning* está sujeito a alguns desafios: o acesso ao conteúdo estudado é lento e intermitente, dispositivos móveis têm poder de processamento modesto e fontes de energias limitadas, a aplicação móvel pode se adaptar a um determinado contexto. Agentes aplicados em dispositivos móveis lidam com muitos destes desafios impostos ao *m-learning*.

Agentes podem realizar download antecipado do conteúdo do estudo baseado no histórico do aprendiz. A qualidade da conexão é usada por agentes para tomar atitudes que diminuam o volume de dados trafegado como, por exemplo, compactar os dados ou não baixar as imagens de uma página HTML. Além disso, o usuário pode executar algumas operações off-line, como enviar uma mensagem mesmo com a conexão indisponível, que serão completadas pelo agente assim que ele tome conhecimento que a rede está novamente acessível. Desta forma, os problemas causados pela intermitência e velocidade da rede são amenizados.

A capacidade de mobilidade dos agentes pode evitar as limitações dos dispositivos móveis. Os agentes móveis migram para um container de agentes em um servidor na rede fixa, realizam um processamento complexo e retornam ao dispositivo móvel com os resultados do processamento.

Agentes em execução dentro do PDA têm acesso a informações de contexto, como localização e carga da bateria. Desta forma, agentes oferecem serviços específicos que atendem às características de um ambiente com mobilidade e modificam seu próprio comportamento com base nestas informações.

Além do *m-learning*, agentes inteligentes podem ser usados em uma grande variedade de aplicações. Agentes são particularmente úteis em aplicações distribuídas envolvendo comunicação ponto-a-ponto. Alguns exemplos de domínios onde sistemas multi-agentes podem ser aplicados incluem aplicações de comércio eletrônico (Ripper et al., 2000), assistentes pessoais para gerenciamento de compromissos (Modi et al., 2004), simuladores (Drogoul & Ferber, 1992) entre outros.

Além das aplicações de agentes no AulaNetM, também são vislumbradas aplicações ou tópicos de pesquisa onde agentes podem ser usados no AulaNet 3.0. Dentre elas, algumas são: o Agente Notificador, o Agente Moderador, o Agente Mediador, o Agente Formador de Grupos e o Agente Tutor. O Agente Notificador poderia ser configurado pelo aprendiz ou mediador para enviar notificações sempre que ocorrer algum evento relacionado ao curso. Este agente poderia, por exemplo, ser configurado para notificar o aprendiz sempre que uma mensagem sua for respondida na conferência. O Agente Moderador seria usado para conduzir a dinâmica de um debate. Um terceiro exemplo seria o do uso de um Agente Mediador, que animaria discussões em conferências. Este agente, por exemplo, enviaria mensagens polêmicas se percebesse um longo período de inatividade na conferência. O Agente Formador de Grupos, baseado em critérios como performance dos aprendizes, conhecimento e grau de afinidade, sugeriria a formação de grupos de trabalho. Finalmente, o Agente Tutor poderia propor conteúdos e cursos relacionados de acordo com o perfil do aprendiz.

Sistemas multi-agentes é um tópico de pesquisa “quente” na área de sistemas de informação (Kinshuk & Lin, 2004) e que, como foi visto, pode trazer vantagens tanto para o AulaNet quanto para o AulaNetM. Desta forma, é desejável que a arquitetura do AulaNet 3.0 forneça suporte ao desenvolvimento de sistemas multi-agentes. Para prover este suporte pode ser usado o framework Jade (Bellifemine et al., 2003), como mostrado na seção a seguir.

5.4.3. O Framework de Agentes Jade

Jade é um framework de aplicação orientada a objeto direcionado para o desenvolvimento de sistemas multi-agentes em conformidade com a especificação

definida pela FIPA (*Foundation for Intelligent Physical Agents*) (FIPA, 2005), uma organização reconhecida pelo IEEE que promove tecnologias baseada em agentes e a interoperabilidade de suas especificações com outras tecnologias (Bellifemine et al., 2005). Jade foi escolhido para mostrar que a arquitetura do AulaNet 3.0 pode ser estendida com outros frameworks pois ele é um framework de agentes que encontra-se em estado de relativa maturidade, é compatível com a especificação da FIPA, é usado comercialmente em algumas empresas como a Telecom Italia LAB, Whitestein Technologies AG e Acklin B.V. (Jade, 2005), é escrito em Java e possibilita o uso de agentes tanto em dispositivos móveis quanto servidores desktops através do pacote de extensão Leap.

Além do framework para desenvolvimento de agentes, o Jade inclui também um ambiente de execução para os agentes, denominado container, e uma ferramenta gráfica onde são realizadas as operações administrativas e o monitoramento do estado dos agentes. Um conjunto de containeres Jade, executando na mesma máquina ou em máquinas distintas, fazem parte de uma plataforma. Agentes em uma mesma plataforma podem se comunicar, opcionalmente usando ontologias. Agentes podem ainda migrar de um container para outro ou clonar-se, gerando um novo agente idêntico (Caire, 2003).

Em uma plataforma, o primeiro container iniciado é denominado o container principal (*Main Container*). Sempre que algum novo container for iniciado na plataforma, ele deve registrar-se neste container. O container principal também é responsável por hospedar os agentes AMS (*Agent Management System*) e DF (*Directory Facilitator*). O primeiro é responsável pelo serviço de nomes que garante que agentes terão nomes únicos em uma plataforma e possibilita a criação e remoção de agentes em outros containeres. O segundo fornece um serviço de páginas amarelas onde agentes podem procurar por outros atentes que prestam serviços necessários para que seus objetivos sejam alcançados. A Figura 5.7 ilustra a execução de duas plataformas de agentes Jade.

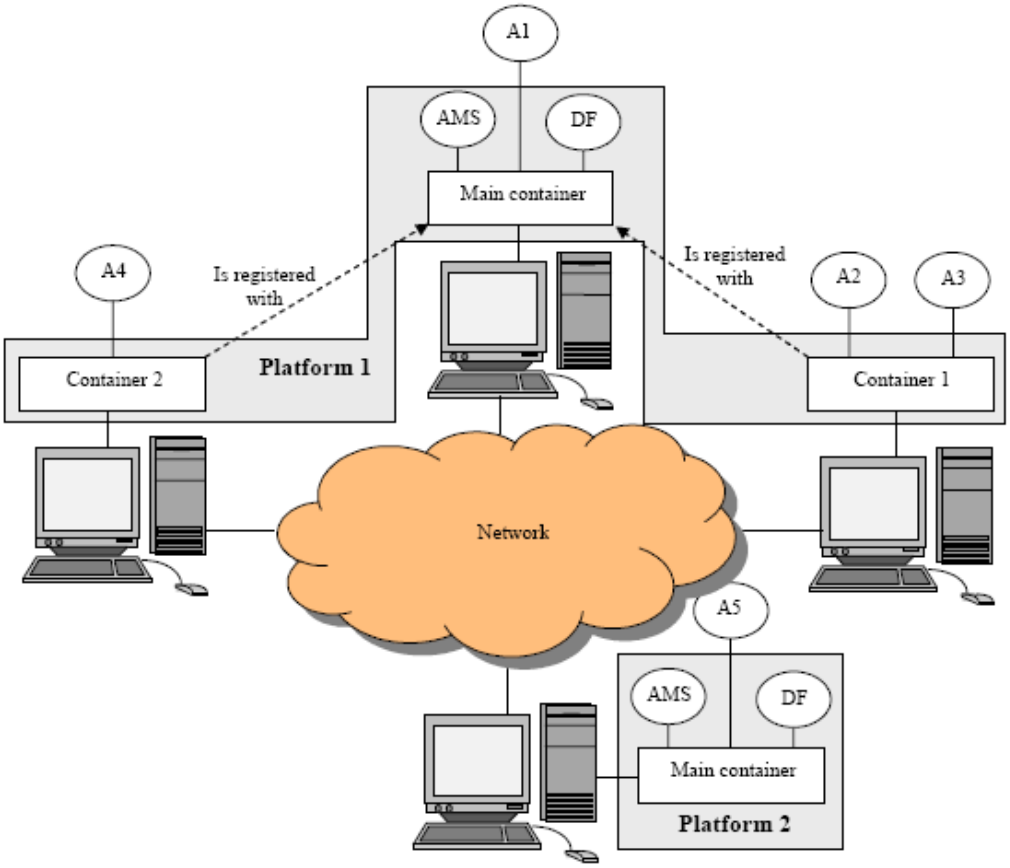

Figura 5.7 – Plataformas de Agentes Jade (Caire, 2003).

A figura mostra duas plataformas. A primeira, denominada Plataform 1, contém três containeres. Os agentes AMS (*Agent Management System*), DF (*Directory Facilitator*) e o agente com o nome A1 se hospedam no container principal. Os agentes A2 e A3 se hospedam no container 1 e o agente A4 se hospeda no container 2. Tanto o container 1 quanto o container 2 se registram no container principal, unindo-se à primeira plataforma. A segunda plataforma possui apenas o container principal, onde se hospedam os agentes AMS, DF e um outro agente chamado A5.

Agentes são construídos estendendo a classe jade.core.Agent. As ações que um agente realiza são especificadas através de comportamentos, classes que estendem jade.core.behaviours.Behaviour ou uma de suas subclasses. Há subclasses específicas para comportamentos que devem ser executados continuamente (CyclicBehaviour), que devem ser executados de tempos em tempos (TickerBehaviour), que são executados apenas uma vez (OneShotBehaviour) e outras. Agentes são identificados pelo seu nome e pela sua plataforma de execução através da classe jade.core.AID.

Com a adição do pacote de extensão Leap, o container Jade pode ser executado em dispositivos móveis e em servidores Java. O pacote Leap substitui partes do núcleo do Jade que passa a se chamar Jade-Leap e possibilita a implantação de containeres Jade-Leap em ambientes J2SE, que é usado em servidores, Personal Java/J2ME CDC, que é usado em PDAs e MIDP, que é usado em telefones celulares (Caire, 2005).

Devido às limitações de hardware em dispositivos como PDAs e telefones celulares, o Jade-Leap possui algumas limitações quando executado nestes ambientes.

No ambiente PJAVA/J2ME CDC, a interface gráfica de administração não pode ser usada, pois ela depende da biblioteca Swing apenas disponível em servidores J2SE. Além disso, não é possível usar os agentes Sniffer e Introspector, utilizados para depuração, em containeres executando no modo stand-alone. O modo de execução stand-alone é explicado mais adiante. Finalmente, os serviços de replicação do container principal, usado para adicionar tolerância à falhas, e de entrega persistente de mensagens, que garante que mensagens são entregues mesmo quando o agente não está disponível, não podem ser usados.

Já no ambiente J2ME MIDP, são impostas as mesmas limitações impostas aos dispositivos PJAVA/J2ME CDC e mais algumas. Não há suporte à mobilidade e à clonagem de agentes nestes ambientes. Comportamentos de agentes que usem *threads* são proibidos. Por fim, o pacote jade.wrapper e os métodos não estáticos da classe jade.core.Runtime não estão disponíveis e algumas classes do pacote de ontologias não funcionam, pois dependem da API de reflexão do Java que não esta disponível na plataforma MIDP.

Jade-Leap pode ser executado em dispositivos móveis de duas formas: no modo *split* e no modo *stand-alone*. No modo *stand-alone* um container Jade-Leap completo é executado no aparelho e no modo *split*, o container é dividido em dois containeres, o *back-end*, executado em um servidor J2SE e o *front-end*, executado no dispositivo móvel. O modo de execução *split* possui inicialização mais rápida e diminui a quantidade de informação trafegada pela rede sem fio. Contudo, não é oferecido suporte à mobilidade e à clonagem de agentes em containeres executando no modo *split*. Desde a versão 3.3 do Jade-Leap o modo de execução *stand-alone* não é mais mantido nem testado e seu uso é desencorajado pelos projetistas do framework (Caire, 2005).

5.4.4.
Jade e a Arquitetura do AulaNet 3.0

Como visto no Capítulo 4, o Spring é o framework responsável por configurar as dependências entre os componentes. Este é um dos principais elementos da camada de negócios do AulaNet 3.0. Se um framework pode ser integrado ao Spring, então este pode ser integrado com sucesso a arquitetura do AulaNet 3.0.

Jade é um framework para a camada de negócios e, portanto, se integrado ao Spring pode ser incorporado a camada de negócios do AulaNet 3.0. Como o Spring não oferece suporte nativo à integração com o Jade, é desenvolvido um adaptador que possibilita a integração de ambos. Este adaptador possibilita que o container Jade seja iniciado dentro de uma aplicação que utiliza o Spring e também que agentes criados com o Jade beneficiem-se dos recursos oferecidos pelo Spring, como por exemplo, a configuração de dependências através de *Dependency Injection*. O diagrama de classes do adaptador é mostrado na Figura 5.8.

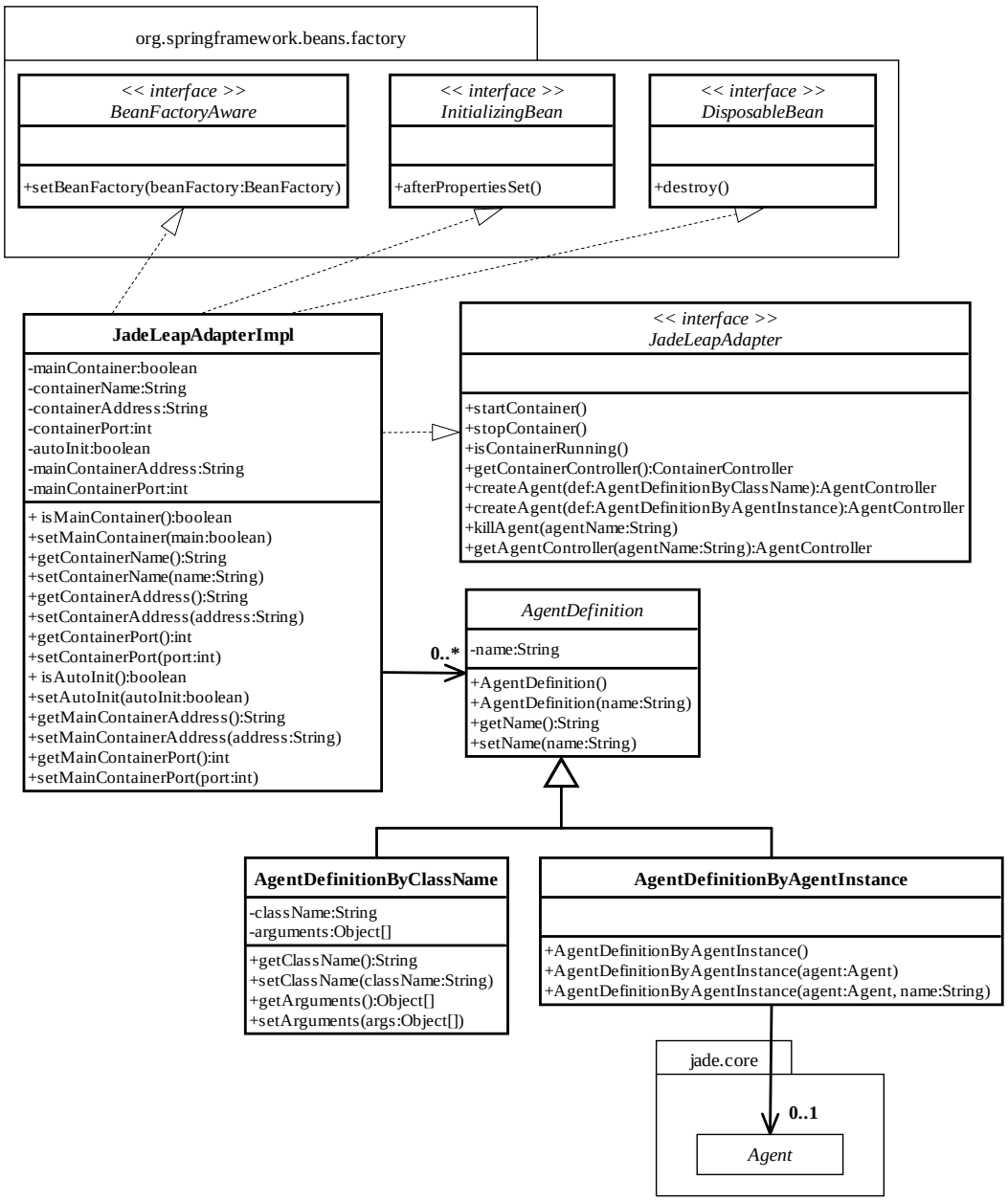

Figura 5.8 – Diagrama de Classes do Adaptador Jade-Leap para o Spring

A classe AgentDefinition e suas subclasses são usadas para definir um agente. Estas definições são usadas pelo adaptador para criar agentes e adicioná-los ao container Jade. A classe abstrata AgentDefinition utiliza apenas o nome do agente para defini-lo, e não é suficiente para adicionar um agente a um container Jade, logo uma de suas subclasses precisa ser utilizada. A subclasse AgentDefinitionByClassName é usada para definir um agente com base no nome da classe que implementa o agente e os argumentos que devem ser passados a ela. A subclasse AgentDefinitionByAgentInstance define um agente através de uma instância da classe jade.core.Agent.

A interface JadeLeapAdapter define os métodos do adaptador que realiza a ponte de integração do Jade-Leap ao Spring. São definidos métodos para iniciar o container Jade (método startContainer()), parar o container (método stopContainer()), verificar se o container está em execução (método isContainerRunning()) e recuperar o controle do container (método getContainerController()) com o qual operações mais complexas podem ser realizadas sobre o container Jade. Além disso, há métodos para criar um agente, segundo a descrição pelo nome da classe ou pela instância (métodos createAgent()), destruir um agente (killAgent()) ou recuperar o controlador do agente usado para executar operações como o acréscimo de comportamentos aos agentes.

A classe concreta que implementa a interface JadeLeapAdapter é a JadeLeapAdapterImpl. Além de implementar a interface JadeLeapAdapter, ela implementa também as interfaces BeanFactoryAware, InitializingBean e DisposableBean do Spring. A primeira define o método setBeanFactory(), que é chamado pelo Spring para passar a fábrica de beans (BeanFactory) responsável pela criação do adaptador. Através desta fábrica, outros beans definidos no Spring podem ser recuperados pelos agentes criados a partir da definição baseada no nome da classe (AgentDefinitionByClassName). Os agentes criados pela definição baseada na instância do agente (AgentDefinitionByAgentInstance) não necessitam da fábrica pois acessam os outros beans definidos no Spring pelo mecanismo de *Dependency Injection*. A interface InitializingBean define o método afterPropertiesSet() chamado quando o adaptador é inicializado e é usada para iniciar o container Jade automaticamente. A interface DisposableBean define o método destroy(), que é chamado pouco antes do adaptador ser destruído, usada para encerrar o container Jade automaticamente.

Além disso, a classe JadeLeapAdapterImpl define propriedades que possibilitam que o modo de execução do Jade seja configurado. A propriedade mainContainer indica se o container a ser executado é um container principal. No caso desta propriedade ser configurada como falsa, as propriedades mainContainerAddress e mainContainerPort devem ser configuradas com o endereço e a porta do container principal. As propriedades containerName, containerAddress e containerPort são usadas para especificar, respectivamente, o nome, endereço e porta do container iniciado pelo adaptador. O vetor

agentDefinitions é configurado com os agentes criados quando o container é iniciado e a propriedade autoInit define se o container é iniciado automaticamente quando o adaptador é criado.

Apesar de possibilitar a integração do Jade ao Spring, o adaptador desenvolvido tem algumas limitações. Ao migrar para outra máquina, as referências que um agente possui para serviços locais tornam-se inconsistentes. O mesmo ocorre com agentes clonados em outras máquinas. Para superar esta limitação, o agente com mobilidade ao retornar para o AulaNet deve comunicar-se com um outro agente sem mobilidade, para que este possa lhe fornecer a referência perdida. Além disso, agentes criados a partir da definição baseada no nome da classe não podem fazer uso do módulo de *Dependency Injection* do Spring, pois eles são instanciados pelo Jade e não pelo Spring. Para contornar este problema o adaptador adiciona a fábrica de beans (BeanFactory) do Spring na última posição do vetor de argumentos passados ao agente. Desta forma, o agente recupera instâncias de beans definidos no Spring.

O adaptador deve ser configurado no arquivo de configuração do Spring, como será visto na seção a seguir. Como nenhuma API específica do Jade ou do JadeLeap é usada na construção do adaptador, este pode ser usado tanto em um quanto no outro. O adaptador não foi desenvolvido para dar suporte ao modo de execução *split* em dispositivos móveis, pois Spring foi desenvolvido para dar suporte apenas aos ambientes J2SE e J2EE, e portanto não funciona em dispositivos móveis. Contudo isto não é empecilho para que um agente em execução em container em um ambiente J2SE ou J2EE com Spring se comunique com outro agente em um dispositivo móvel MIDP ou PJAVA/J2ME CDC, como é visto na próxima seção.

5.4.5. Prova de Conceito com Agente Móvel

Para testar a integração entre o Jade e o Spring através do adaptador, bem como a viabilidade do uso de agentes em dispositivos móveis se comunicando com outros agentes em servidores, desenvolveu-se uma prova de conceito. Esta prova é dividida em duas partes. Na primeira parte, é mostrado o sistema multi-agentes “O Que Há de Novo na Conferência?” que tem pouca aplicação prática,

mas que é mostrado com mais detalhes por ser mais simples, e como conseqüência mais didático. Na segunda parte é mostrado um sistema multi-agentes “Assistente do Mediador” que tem mais aplicação prática, pois é construído para superar alguns problemas reportados por usuários do AulaNetM. Este não é mostrado com tantos detalhes devido a sua complexidade.

5.4.2.1. MAS 1: O Que Há de Novo na Conferência?

Esta aplicação é usada para obter atualizações sobre uma determinada conferência. A cada solicitação do usuário do PDA, a aplicação informa as mensagens novas, que foram removidas, que tiveram suas categorias alteradas e que foram avaliadas desde a última consulta do usuário.

Para implementar esta aplicação são utilizados dois agentes. O primeiro, denominado Agente Móvel (MobileAgent), é hospedado por um container Jade-Leap em execução em um PDA iPAQ 5555, utilizando a máquina virtual J2ME CDC CrEme (2005). O modo de execução é o *split* e o container neste dispositivo atua como *front-end*. O segundo agente, denominado Agente Conferência (ConferenceAgent), é hospedado por um container Jade-Leap em execução no servidor J2EE JBoss em conjunto com o protótipo do serviço Conferência desenvolvido a partir da arquitetura definida para o AulaNet 3.0. Este container funciona como o *back-end*. O container principal da plataforma é executado separadamente, para facilitar a depuração do sistema. A Figura 5.9 ilustra a interação entre os containeres.

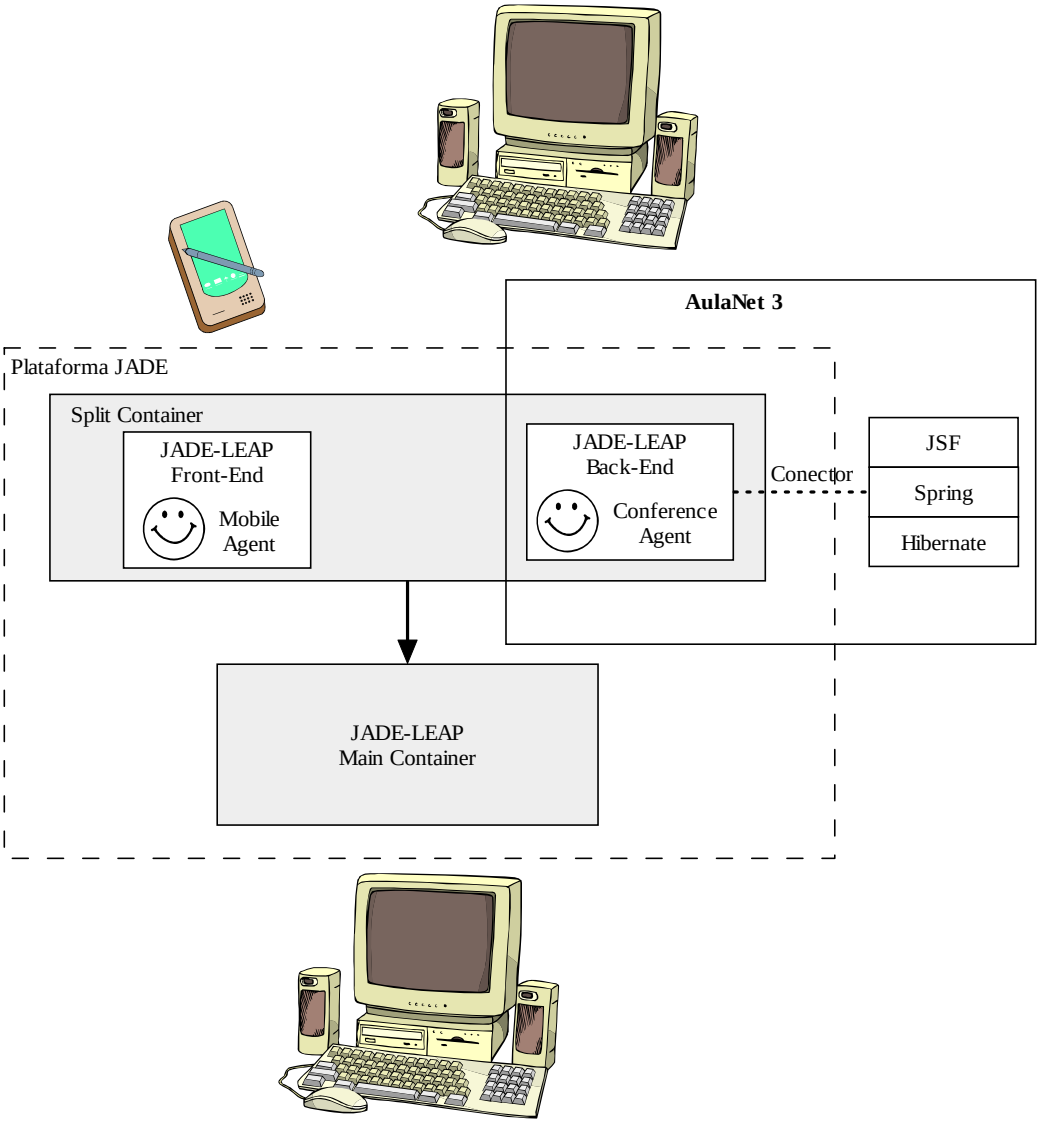

Figura 5.9 – Plataforma de Execução do MAS 1

O Agente Móvel, quando acionado pela aplicação, se comunica com o Agente Conferência para buscar atualizações sobre uma determinada conferência. O Agente Conferência então acessa a camada de negócios do AulaNet 3.0, recuperando o estado dela e envia de volta ao Agente Móvel a lista das mensagens novas, das mensagens que foram removidas, das mensagens que tiveram sua categoria alterada e das mensagens que foram avaliadas desde a última consulta do Agente móvel.

O Agente Conferência é implementado da mesma forma que qualquer outro agente seria implementado na plataforma JadeLeap. A única diferença é que,

como ele é executado em conjunto com o Spring, obtém os benefícios proporcionados por este framework. A Listagem 5.1 mostra o código do agente.

```
01: public class ConferenceAgent extends Agent {

02:     private ConferenceFacade facade;

03:     protected void setup() {
04:         addBehaviour(new WhatIsNewInConferenceBehaviour());
05:     }

06:     protected void takeDown() {
07:         facade = null;
08:     }

09:     public final ConferenceFacade getConferenceFacade() {
10:         return conferenceFacade;
11:     }

12:     public final void setConferenceFacade(ConferenceFacade cf) {
13:         facade = cf;
14:     }
15: }
```

Listagem 5.1 – Agente Conferência

Na linha 01, a classe do agente estende a classe jade.core.Agent. Todos os agentes do Jade devem estender esta classe. Na linha 02 é declarada a variável para o *Façade* da conferência. O método setup(), chamado pelo container Jade para configurar o agente quando este é adicionado ao container, é declarado entre as linhas 03 e 05. Na linha 04 o comportamento WhatIsNewInConferenceBehaviour, que é descrito mais adiante, é acrescentado ao agente. O método takeDown(), chamado quando o agente é removido do container, é declarado no trecho entre as linhas 06 e 08. No trecho entre as linhas 09 e 14 são declarados os métodos de acesso para a propriedade *facade*. Esta propriedade é configurada pelo Spring por *Dependency Injection*.

O comportamento WhatIsNewInConferenceBehaviour é responsável por responder ao Agente Móvel o que há de novo na conferência. O código fonte deste comportamento é listado a seguir.

```
16: public class WhatIsNewInConferenceBehaviour extends CyclicBehaviour {

17:     private final Map lastConference = new HashMap();

18:     public void action() {
19:         MessageTemplate mt = MessageTemplate.and(
20:             MessageTemplate.MatchPerformative(ACLMessage.QUERY_IF),
21:             MessageTemplate.MatchConversationId("conference-status"));

22:         ACLMessage msg = myAgent.receive(mt);
23:         if (msg != null) {
24:             AID sender = msg.getSender();
25:             Conference oldConference = (Conference) lastConference.get(sender);
26:             String conferenceId = msg.getContent();
27:             ConferenceFacade ca = (ConferenceAgent) myAgent;
28:             ConferenceFacade facade = ca.getConferenceFacade();
29:             Conference newConference = facade .findConferenceById(conferenceId);

30:             List newMessages = getNewMessages(oldConference, newConference);
31:             List deletedMessages = getDeletedMessages(oldConference, newConference);
32:             List categorizedMessages = getCategorizedMessages(oldConference, newConference);
33:             List evaluatedMessages = getEvalutatedMessages(oldConference, newConference);
34:             String content = buildReplyContent(newMessages, deletedMessages,
35:                 categorizedMessages, evaluatedMessages);

36:             ACLMessage reply = msg.createReply();
37:             reply.setContent(content);
38:             myAgent.send(reply);
39:             lastConference.put(sender, newConference);
40:         } else {
41:             block();
42:         }
43:     }
44: }
```

Listagem 5.2 – Comportamento WhatIsNewInConferenceBehaviour

Na linha 16 o comportamento é declarado como CyclicBehaviour, o que significa que ele é executado indefinidamente. A variável lastConference é um mapa que guarda um espelho das últimas consultas dos Agentes Móveis. No trecho entre as linhas 18 e 21 é declarado um template de mensagem, que indica o formato da mensagem que o agente espera receber. Na linha 22, uma mensagem em conformidade com o template é solicitada. O remetente da mensagem é recuperado na linha 24 e a partir dele é recuperada o status da conferência em sua última consulta na linha 25. A partir do id da conferência recuperado na linha 26, o estado atual da conferência é recuperado através do *Façade* na linha 29. Os comandos no trecho entre as linhas 30 e 35 chamam métodos que calculam as mensagens novas, as removidas, as que tiveram sua categoria modificada e as que foram avaliadas e depois compõem a resposta que será enviada. Estes métodos não são exibidos por motivos de clareza.

Na linha 36 a resposta é criada. O conteúdo dela é configurado na linha 37 e ela é enviada na linha 38. Na linha 39 o mapa das conferências passadas é atualizado.

Para que a dependência do agente com o *Façade* da conferência seja satisfeita, é preciso que o arquivo de configuração do Spring seja configurado apropriadamente. É também através do arquivo de configuração do Spring que o adaptador do Jade-Leap é configurado. A Listagem 5.3 mostra o arquivo de configuração do Spring.

```
45: <beans>
46:    <bean id="jadeContainerConnector" class="JadeLeapConnectorImpl">
47:       <property name="mainContainer">
48:          <value>false</value>
49:       </property>
50:       <property name="autoInit">
51:          <value>true</value>
52:       </property>
53:       <property name="containerName">
54:          <value>SpringBackEnd</value>
55:       </property>
56:       <property name="containerPort">
57:          <value>2999</value>
58:       </property>
59:       <property name="mainContainerAddress">
60:          <value>127.0.0.1</value>
61:       </property>
62:       <property name="mainContainerPort">
63:          <value>1999</value>
64:       </property>
65:       <property name="agentDefinitions">
66:          <list>
67:             <bean class="AgentDefinitionByAgentInstance">
68:                <property name="name">
69:                   <value>Conference-Agent</value>
70:                </property>
71:                <property name="agentInstance">
72:                   <bean class="ConferenceAgent">
73:                      <property name="facade">
74:                         <ref bean="conferenceFacade"/>
75:                      </property>
76:                   </bean>
77:                </property>
78:             </bean>
79:          </list>
80:       </property>
81:    </bean>
82: </beans>
```

Listagem 5.3 – Configuração no Spring do Adaptador Jade-Leap

O adaptador do Jade-Leap é configurado no trecho entre as linhas 45 e 81. Por motivos de clareza o bean conferenceFacade não é exibido na listagem. No trecho entre as linhas 47 e 49 a propriedade mainContainer é definida como false, indicando que o container executado não é principal e entre as linhas 50 e 52 a propriedade autoInit é configurada como true, o que indica que o container é inicializado automaticamente. O nome do container é declarado como SpringBackEnd no trecho entre as linhas 53 e 55 e a porta em que o container executa é declara como 2999 no trecho entre as linhas 56 e 58. Entre as linhas 59 e 61 o endereço do container principal é configurado para 127.0.0.1 e entre as linhas 62 e 64 a porta é configurada para 1999. No trecho entre as linhas 65 e 80, a definição dos agentes é declarada. Neste exemplo, apenas o Agente Conferência é configurado.

Na linha 67 é declarado que a definição de agentes usada é a definição por instância de agente. Desta forma, é possível usar a funcionalidade de *Dependency Injection* do Spring. O nome do agente, ConferenceAgente, é declarado na linha 69 e a instância do agente é declarada entre as linhas 72 e 76. A classe do agente é definida na linha 72 e a dependência com o *Façade* da conferência é configurada no trecho entre as linhas 73 e 75.

O Agente Móvel é implementado usando J2ME CDC e, portanto, não usa o Spring e seu adaptador para o Jade-Leap. Seu código é exibido na Listagem 5.4. O Código responsável pela criação da interface no PDA é omitido por questões de clareza.

```
83: public class MobileAgent extends Agent {

84:     protected void setup() {
85:         System.out.println("Agent created.\n");
86:     }
87: }
```

Listagem 5.4 – Agente Móvel

O código do Agente Móvel é pequeno, mas é necessário, pois a classe Agent é abstrata e não pode ser instanciada. Tudo que esta classe faz é imprimir uma mensagem indicando que o agente foi criado. O comportamento do agente é

implementado na classe ConferenceQueryBehaviour, responsável pela comunicação com o Agente Conferência. Este comportamento, cujo código é mostrado na Listagem 5.5, é adicionado ao agente dinamicamente quando um usuário aciona o botão de atualização.

```
088: public class ConferenceQueryBehaviour extends Behaviour {

089:    private int step = SEND_QUERY;
090:    private static final int SEND_QUERY = 1;
091:    private static final int RECEIVE_RESPONSE = 2;
092:    private static final int FINISH = 3;
093:    private String conferenceId;

094:    public ConferenceQueryBehaviour(String conferenceId) {
095:       this.conferenceId = conferenceId;
096:    }

097:    public void action() {
098:       switch (step) {
099:       case SEND_QUERY:
100:          AID receiver = new AID("Conference-Agent", AID.ISLOCALNAME);
101:          ACLMessage query = new ACLMessage(ACLMessage.QUERY_IF);
102:          query.addReceiver(receiver);
103:          query.setConversationId("conference-status");
104:          query.setContent(conferenceId);
105:          query.setReplyWith("query" + System.currentTimeMillis());
106:          myAgent.send(query);

107:          template = MessageTemplate.and(
108:             MessageTemplate.MatchConversationId("conference-status"),
109:             MessageTemplate.MatchInReplyTo(query.getReplyWith()));
110:          step = RECEIVE_RESPONSE;
111:          break;
112:       case RECEIVE_RESPONSE:
113:          ACLMessage reply = myAgent.receive(template);

114:          if (reply != null) {
115:             System.out.println(reply.getContent());
116:             step = FINISH;
117:          } else {
118:             block();
119:          }
120:          break;
121:       }
122:    }

123:    public boolean done() {
124:       return step == FINISH;
125:    }
126: }
```

Listagem 5.5 – Comportamento ConferenceQueryBehaviour

Como é visto na linha 088 a classe ConferenceQueryBehaviour, estende a classe Behaviour que se encontra no topo da hierarquia das classes que oferecem suporte à construção de comportamentos. Esta classe é útil para a construção de comportamentos que funcionam como máquina de estados. O trecho entre as linhas 097 e 122 declara o método action, executado toda vez que o comportamento é chamado e o trecho entre as linhas 123 e 125 declara o método done, que testa se o comportamento deve ser chamado novamente.

A variável que armazena o estado é declarada na linha 089. Constantes representando os estados enviar pergunta (SEND_QUERY), receber resposta (RECEIVE_RESPONSE) e final (FINISH) são declaradas entre as linhas 090 e 092. Na linha 093 é declarada a variável que armazena o id da conferência para a qual serão obtidas as atualizações.

Um construtor que recebe o id da conferência como parâmetro é declarado no trecho entre as linhas 094 e 096. O estado enviar pergunta é delimitado pelas linhas 099 e 111. O endereço do destinatário é criado na linha 100 e a mensagem é criada na linha 101. O destinatário é adicionado à mensagem na linha 102 e o identificador da conversa é configurado na linha 103. O id da conferência para a qual se deseja obter atualizações é adicionado como conteúdo da mensagem na linha 104. Na linha 105, um número único é acrescentado a mensagem, para evitar o tratamento de mensagens falsas. A mensagem é enviada na linha 106 e no trecho entre as linhas 107 e 109 um template de mensagem com o formato da resposta aguardada é criado. O estado enviar pergunta é delimitado pelas linhas 112 e 120. A mensagem é recebida na linha 113 e seu conteúdo é exibido na tela (linha 115).

A interface gráfica desenvolvida para o PDA possui 6 botões divididos em duas linhas. Na primeira linha encontram-se os botões para iniciar o container Jade, parar o container e ativar o agente (acrescentar o comportamento ConferenceQueryBehaviour a ele). Na segunda linha encontram-se os botões para limpar a tela, configurar a aplicação e fechar o programa. A Figura 5.10 e a Figura 5.11 mostram o programa em execução.

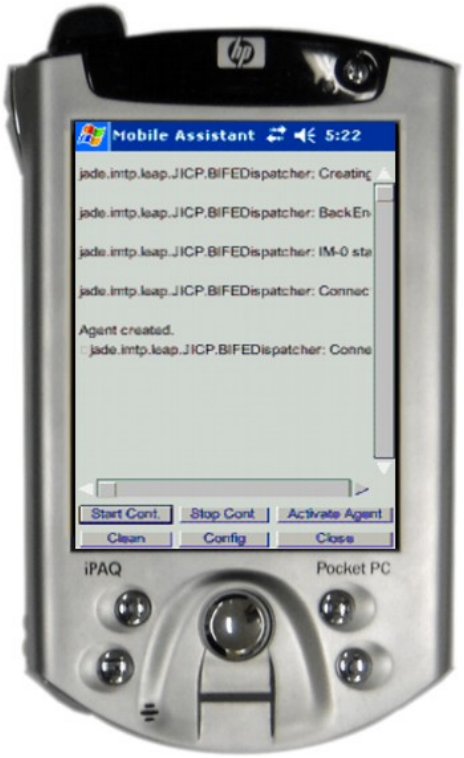

Figura 5.10 – PDA Após a Inicialização do Jade-Leap

Figura 5.11 – PDA Após a Primeira Consulta

A Figura 5.10 mostra a aplicação após o botão “Start Cont.” ser clicado, resultando no início do container Jade. Já na Figura 5.11, é mostrado o estado da aplicação após o botão “Activate Agent”, responsável pela ativação do agente, ser acionado. Como é a primeira consulta realizada, todas as mensagens da conferência são retornadas como mensagens novas. A Figura 5.12 mostra uma nova ativação do agente.

Figura 5.12 - PDA Após a Segunda Consulta

Como a segunda ativação é realizada logo em seguida da primeira, sem que haja mudança na conferência, o agente não detecta alterações. Este exemplo, apesar de didático, possui pouca aplicação prática já que não apresenta nenhuma evolução dos serviços já existentes no AulaNet e AulaNetM. Na próxima seção é vista a segunda parte da prova de conceito, com uma aplicação baseada em um problema real encontrado por mediadores do AulaNet.

5.4.2.2. MAS 2: Alerta de Condições Indesejáveis

A aplicação desenvolvida nesta segunda parte tem como objetivo alertar o mediador de um curso quando uma condição indesejável estiver ocorrendo no serviço Conferências. São verificadas quatro condições que normalmente são indesejáveis. A primeira refere-se ao seu nível de atividade, o que gera uma notificação caso seja detectada a ausência de mensagens em um intervalo de tempo maior do que um período previamente estabelecido. A segunda informação verifica se uma questão está sendo pouco debatida ou se está polarizando a conferência em detrimento das outras. Neste caso, o mediador é notificado caso alguma mensagem categorizada como “Questão” possua uma quantidade de

respostas menor ou maior que um limite pré-configurado. Isto só é feito a partir de 12 horas após o início da conferência, dando tempo para os aprendizes iniciarem a discussão. A terceira informação refere-se ao nível de atividade dos aprendizes: uma notificação é gerada quando um aprendiz apresentar uma participação que possa ser considerada muito baixa segundo os critérios do curso. A quarta e última informação disponibiliza uma medida para o mediador estimar o nível de interatividade da conferência: por exemplo, uma notificação é gerada toda vez que a porcentagem de folhas da estrutura da árvore da conferência for maior que 50%, ou seja, quando mais da metade das mensagens não estiverem sendo respondidas. Devido às diferentes características das turmas e dos propósitos das conferências, os valores dos parâmetros para que as notificações sejam acionadas devem ser configurados pelos mediadores para cada conferência através de uma interface específica para este fim.

Ao receber estas notificações os mediadores podem tomar decisões que mudem o rumo da conferência, contornando as situações indesejáveis. O AulaNetM provê meios para a verificação destas situações através de informações visuais (Figura 5.13) e estatísticas (Figura 5.14), porém, o processo de detecção das condições indesejáveis não é automatizado, isto é, os Mediadores precisam acessar diretamente o AulaNetM para obter a informação. Conseqüentemente, na maior parte dos casos os mediadores tomam conhecimento da condição tardiamente. Além disso, mediadores entrevistados reclamaram da dificuldade de estabelecer conexão e de mantê-la uma vez estabelecida. Pesquisas revelaram indícios de que os mediadores passavam mais tempo tentando conectar-se ou reconectar-se do que analisando a conferência.

Figura 5.13 – Informações Visuais no AulaNetM

Figura 5.14 – Informações Estatísticas no AulaNetM

Para contornar estes problemas uma MAS foi desenvolvido. Os mediadores não precisam mais monitorar o estado da conferência, pois os agentes já fazem isto e o mecanismo de tolerância a falhas presente no Jade possibilita que mensagens não entregues devido a falhas na conexão sejam adiadas e entregues ao retomar a conexão de forma transparente ao usuário. A aplicação consiste em um sistema multi-agentes constituído por dois tipos de agentes diferentes: o Conference Agent e o Mediator Assistant.

O Conference Agent é o mesmo descrito na seção anterior, mas a ele é adicionado os comportamentos que permitem detectar as condições indesejáveis na conferência. O Conference Agent é pró-ativo e reativo, e permanece monitorando a Conferência do AulaNet na tentativa de detectar situações indesejáveis. Como cada curso no AulaNet pode ter várias conferências, cada nova conferência ganha um Conference Agent na sua ativação e, na sua desativação, este agente é destruído.

O Mediator Assistant é executado dentro de um container Jade-Leap em execução dentro de um PDA iPAQ 5555, o mesmo ambiente utilizado na aplicação da seção anterior. Este agente permanece em execução no PDA initerrupdamente, esperando por mensagens do Agente Conference Agent. Ao receber uma mensagem, o agente Mediator Assistant emite um alerta, comunicando ao seu usuário a situação indesejada ocorrida. Este agente é

autônomo, e pode optar por não mostrar um alerta em determinados casos, como a desabilitação explícita de um determinado tipo de alerta pelo mediador ou a excessiva repetição do alerta. As figuras Figura 5.15 e Figura 5.16 mostram o Mediator Assistant em execução.

Figura 5.15 – Informações Visuais no AulaNetM

Figura 5.16 – Informações Estatísticas no AulaNetM

A Figura 5.15 mostra a tela do agente Mediator Assistant e a Figura 5.16 mostra o agente após receber vários alertas. Esta aplicação é capaz de alertar os mediadores tão logo a condição seja detectada e assim, estes são capazes de tomar atitudes mais cedo para contornar a condição indesejável com mais rapidez.

5.5. Conclusão

A arquitetura do AulaNet 3.0 pode ser vista através de duas perspectivas: a da arquitetura de aplicação, que apresenta uma visão mais alto nível e a da arquitetura técnica, que apresenta uma visão mais baixo nível.

Quando contemplada do ponto de vista da arquitetura de aplicação, a arquitetura do AulaNet 3.0 apresenta dois níveis de componentização: o dos serviços e o dos componentes de colaboração com os quais os serviços são construídos. A arquitetura de aplicação apresenta dois frameworks de componentes: o Service Component Framework, que provê os contratos que os serviços de groupware devem implementar e o Collaboration Component Framework, que fornece os contratos para criar componentes 3C com os quais os serviços de groupware são construídos. Tanto os serviços quanto os componentes 3C são desenvolvidos segundo a arquitetura técnica.

A arquitetura técnica segue uma abordagem em três camadas e também o padrão MVC (Fowler, 2002). A camada de recursos relaciona os recursos necessários para à execução do AulaNet, por exemplo o banco de dados relacional. A camada de negócios é onde a lógica da aplicação é implementada. Nela encontram-se *Data Transfer Objects* (DTOs) (Fowler, 2005), classes do modelo que representam entidades de negócio e são usadas para transportar dado entre camadas (correspondem ao M do MVC), *Data Access Objects* (DAOs) (Alur et al., 2001), classes que encapsulam o acesso ao banco de dados e *Façades* (Gamma et al., 1995), classes que servem como ponto de entrada para a camada de negócios. A camada de apresentação, que expõe a lógica de negócios ao usuário, é composta pelo controlador, o C do MVC além de páginas JSP, que correspondem ao V do MVC. Ao longo desta dissertação foram vistos como vários frameworks podem ser acrescentados proporcionando uma base de serviços de infra-estrutura. Após o acréscimo destes frameworks, a arquitetura é descrita segundo a Figura 5.17.

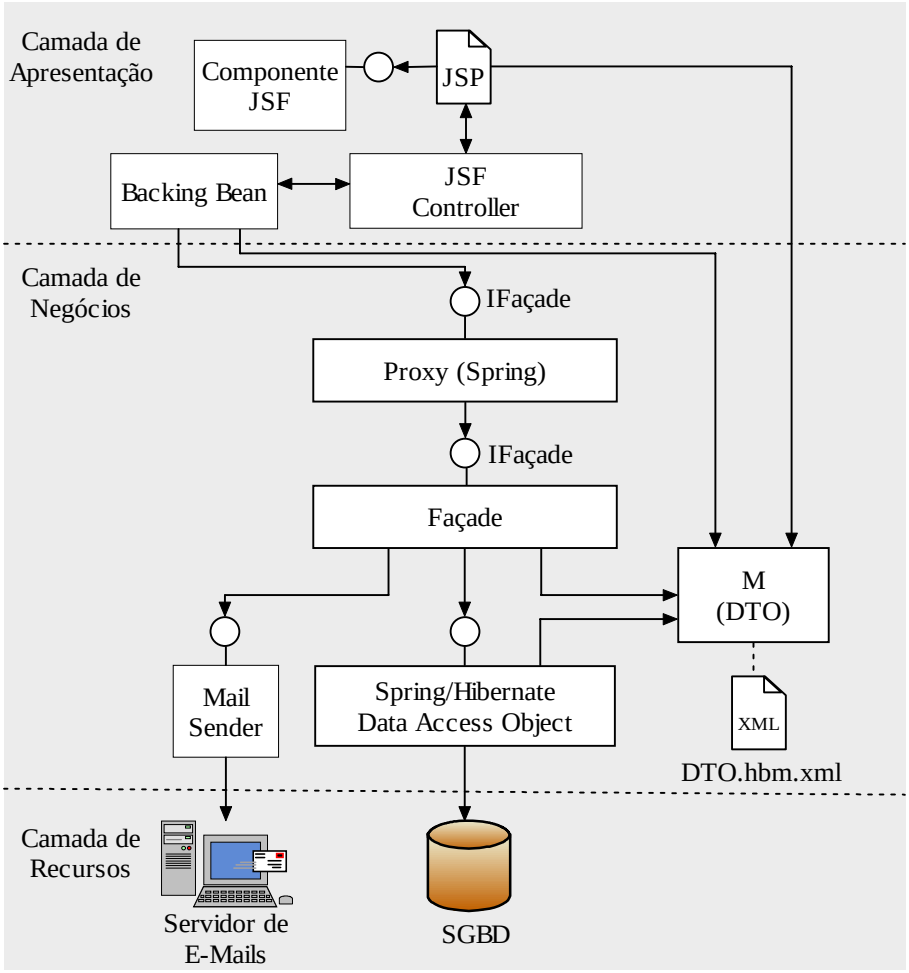

Figura 5.17 – Arquitetura Técnica do AulaNet 3.0 com Frameworks

Com o crescimento do uso de dispositivos móveis e com o advento das redes sem fio, espera-se que seja possível ter acesso a informações, comunicação e serviços em qualquer lugar e a qualquer instante. Com o objetivo de investigar o uso de equipamentos móveis na aprendizagem colaborativa, iniciou-se o desenvolvimento de uma extensão do ambiente AulaNet para dispositivos móveis denominada AulaNetM.

Atualmente o AulanetM integra-se ao AulaNet 2.1 no nível de dados. A arquitetura do AulaNet 3.0 foi projetada para possibilitar a integração em nível de serviços, proporcionando um maior reuso. A integração pode ser alcançada através da técnica reposição da camada de apresentação ou da exposição de serviços remotamente.

Pela técnica da reposição da camada de apresentação, a camada de negócios de negócios é totalmente reaproveitada enquanto que a camada de apresentação é

substituída por uma outra mais adequada ao dispositivo móvel, utilizando HTML ou WML. Esta técnica é adequada a clientes magros e sua principal vantagem é a maior facilidade que equipes de um projeto tendem a encontrar ao participar de outro, já que ambos usam as mesmas interfaces de negócios. A maior desvantagem é que clientes magros não têm acesso direto a informações como a localização do PDA, por exemplo, dificultando a implementação de serviços específicos de mobilidade de forma transparente.

A técnica da exposição de serviços remotos é implementada através da adição de *Proxys* do Spring que exportam os serviços remotamente. Clientes magros ou gordos acessam os serviços do AulaNet através destes *Proxys*. A principal vantagem desta técnica é que clientes gordos têm acesso às informações do dispositivo, como por exemplo, a localização. Desta forma, podem dar suporte a serviços específicos de mobilidade de forma transparente. A principal desvantagem é que as chamadas remotas são mais complexas de serem feitas e oferecem desempenho inferior. Apesar de prover suporte tanto para clientes magros quanto gordos, esta técnica é mais adequada para os gordos.

Apesar das inegáveis vantagens do uso de dispositivos móveis, o desenvolvimento de sistemas nestes ambientes oferece desafios devido à baixa capacidade de processamento do aparelho, conexão intermitente e de baixa qualidade, bateria que precisa ser recarregada, etc. Segundo Kinshuk & Lin (2004), sistemas de agentes inteligentes têm um grande potencial para solucionar estes desafios.

Além das aplicações de agentes para dispositivos móveis, há várias outras possíveis para o AulaNet 3.0. Por isto, decidiu-se que a arquitetura do AulaNet 3.0 deveria possibilitar o uso de agentes. Para evidenciar que a arquitetura do AulaNet 3.0 pode ser estendida com outros frameworks e que pode dar suporte ao uso de agentes, decidiu-se investigar a integração do AulaNet ao framework de agentes Jade.

Para que um framework seja integrado a arquitetura do AulaNet, é preciso que este seja compatível com o Spring. Como o Spring não oferece suporte nativo de integração com o Jade, foi necessário criar um adaptador. Através do adaptador, podem ser realizadas operações sobre o container Jade e sobre agentes. Como os agentes são instanciados pelo Spring, eles podem fazer uso das

funcionalidades proporcionadas por ele, incluindo *Dependency Injection* com outras classes.

Para testar a integração do Jade com o Spring bem como a viabilidade do uso de agentes em dispositivos móveis foi realizada uma prova de conceito em duas etapas. Ambas envolveram agentes em ambientes móveis comunicando-se com agentes no AulaNet.

O teste de conceito foi implementado com sucesso e com ele, constatou-se que a arquitetura do AulaNet 3.0 pode ter outros frameworks não previstos inicialmente incorporados, desde que eles sejam compatíveis com o Spring ou possam ser adaptados através de um adaptador para uso com o Spring. Constatou-se também que é possível desenvolver agentes em dispositivos móveis usando Jade e que eles podem se comunicar com outros agentes localizados em servidores.